Theoretical Computer Science 95 (1992) 245–261
Elsevier

On-line computation of minimal and maximal length paths*

Giorgio Ausiello

Dipartimento di Informatica e Sistemistica, via Eudossiana 18, 00184 Rome, Italy

Giuseppe F. Italiano

Dipartimento di Informatica e Sistemistica, via Eudossiana 18, 00184 Rome, Italy, and Department of Computer Science, Columbia University, New York, NY 10027, USA

Alberto Marchetti Spaccamela

Dipartimento di Matematica Pura e Applicata, Università di L'Aquila, L'Aquila, Italy

Umberto Nanni

Dipartimento di Informatica e Sistemistica, via Eudossiana 18, 00184 Rome, Italy, and Dipartimento di Matematica Pura e Applicata, Università di L'Aquila, L'Aquila, Italy

Communicated by M. Nivat
Received December 1987
Revised February 1990

Abstract

Ausiello, G., G.F. Italiano, A. Marchetti Spaccamela and U. Nanni, On-line computation of minimal and maximal length paths, Theoretical Computer Science 95 (1992) 245–261.

We consider the problem of maintaining *minimum* length paths in a directed graph $G=(V,E)$ with n nodes while inserting new arcs. A data structure which supports the following operations is presented: an *add* operation, which inserts an arc in the digraph, and a *minpath* operation, which returns a minimal length path between a pair of nodes. The data structure supports each minpath operation in $O(k)$ worst case time, where $k<n$ is the length of the returned path; moreover, if we assume that the weights of the arcs are integer numbers in the range $[1...W]$, then the expected cost of any sequence of add operations is $O(\min(n^4, n^3 \max(W, \log n)))$ time. The space complexity is $O(n^2)$. The same algorithm can be used for solving the problem of maintaining *maximum* length paths when the digraph is acyclic and add operations preserve acyclicity.

* Work partially supported by the Italian MPI National Project "Algoritmi e Strutture di Calcolo" and by the Commission of the European Communities under the ESPRIT Basic Research Project 3075 "ALCOM". The second author was partially supported by NSF Grants CCR-86-05353, CCR-88-14977 and by an IBM Graduate Fellowship. The fourth author was partially supported by SELENIA S.p.A.

1. Introduction

Significant progress has been recently made in the study of dynamic data structures on graphs, i.e. data structures that, in addition to queries on the graph, support insertions and deletions of arcs [7, 9, 12, 13, 17, 18]. In particular, the problem of determining minimum or maximum cost paths between pairs of nodes in graphs, where arcs may be dynamically inserted, is relevant in many applications, such as all those related to network design and management [8]. A different application concerns the representation of transitive implications in semantic models and semantic networks [19], a context where finding a path between two nodes (concepts) is a common operation in deriving semantic connections between concepts. A particular case of this application concerns database modelling [3] and problem solving [16].

The problem discussed in this paper can be formalized as follows. Suppose we are given a directed finite graph $G=(V, E)$ to which arcs may be added, one at a time; each arc has an integer weight in the range $[1 \ldots W]$. We require that questions of the type:

"*Which is the shortest path between nodes u and v?*"

may have to be answered at any time in an "on-line" fashion. The naive algorithm which checks the minimal path for each question separately takes time $O(q|E|)$ in answering q questions.

In this paper an algorithm is presented for maintaining a data structure in which each question is answered in $O(k)$ time, where k is the number of arcs of the achieved path, and for which the total expected time involved in maintaining the data structure when $O(n^2)$ arcs are successively added is $O(\min(n^4, n^3 \max(W, \log n)))$, where W is the maximum arc weight. As a subsidiary result, we show how maximal length paths can be retrieved by means of *maxpath* operations if the insertion of arcs allows the digraph to remain acyclic. This is not a restriction since the maximal length path problem on graphs with cycles is known to be NP-complete [10].

A somewhat simpler problem was tackled by Ibaraki and Katoh [12], who introduced an algorithm for updating the transitive closure of a graph, which enables one to answer connectivity questions ("is a node v reachable from a node u?") in constant time and maintains the transitive closure when m arcs are successively added to $G=(V, E)$ in $O(n^3)$ time. In [13] a data structure supporting path retrieval operations and insertions of arcs both in $O(n)$ amortized time was introduced. The achieved path was arbitrarily chosen, while in this paper we want to restrict ourselves to paths of minimal or maximal length, i.e. paths with a minimal or maximal number of arcs.

Even and Gazit [6] and Rohnert [17] considered the more general problem of updating the solution of the All Pairs Shortest Path Problem between two successive modifications of the cost function. The particular case of unit costs seems not to introduce any improvement to the bounds proposed in those papers, for which a single arc insertion can require $O(n^2)$ worst-case time. This gives a total time of

$O(n^4)$ when $O(n^2)$ arcs are inserted. On the other hand, solving the All Pairs Shortest Path Problem from scratch requires $O(n^2 \log n)$ expected running time (and, therefore, $O(n^4 \log n)$ total time over $O(n^2)$ arc insertions) as shown by Moffat and Takaoka [15]. Currently, no efficient dynamic solution is known when the expected time complexity is taken into account. As a result, our algorithm favorably compares to the previous known bounds in case of unit edge costs.

The remainder of the paper consists of five sections. In Section 2 graph terminology is introduced. A description of the data structure is given in Section 3, while in Section 4 we analyze its expected time complexity. Section 5 deals with maximal length paths. Finally, Section 6 contains some concluding remarks.

2. Preliminary definitions and notation

We assume that the reader is familiar with the standard graph terminology as contained in [1, 11, 20]. In particular, a *directed graph* $G=(V, E)$ (sometimes called a *digraph*) is a finite set $V=\{1, 2, \ldots, n\}$ of *nodes* and a finite set E of *arcs* such that each arc e has a *head* $h(e) \in V$ and a *tail* $t(e) \in V$. We consider the arc e as leading from $t(e)$ to $h(e)$ and we say that the arc e leaves $t(e)$ and enters $h(e)$. An integer weight $w(i, j)$ is associated with arc (i, j).

A *path* $p=e_1, e_2, \ldots, e_k$ is a sequence of arcs such that $h(e_i)=t(e_{i+1})$ for $1 \leqslant i \leqslant k-1$. The path is from $t(p)=t(e_1)$ to $h(p)=h(e_k)$ and contains arcs $e_1, e_2, \ldots, e_k$ and nodes $t(e_1), t(e_2), \ldots, t(e_k), h(e_k)$. The path is *simple* if all its nodes are distinct. The *length* of a path is the sum of the weights of the arcs it contains. As a special case, a single node denotes a path of length 0 from itself to itself. A *cycle* is a nonempty path from a node to itself. A node v is *reachable* from a node u if there is a path from u to v: in such a case u is said to be an *ancestor* of v and v a *descendant* of u. If, in addition, $u \neq v$, u is a *proper ancestor* of v and v is a *proper descendant* of u. If there is an arc from u to v, v is *adjacent* to u. If $G=(V, E)$ is a digraph, the digraph which has the same vertex set as G but has an arc from u to v if and only if there is a path from u to v in G, is called the *transitive closure* of G. A digraph with no cycles is called a *directed acyclic graph* (*dag*). A *rooted tree* is a dag satisfying the following properties:

(i) there is only one node, called the *root*, which no arcs enter;

(ii) every node except the root has exactly one entering arc;

(iii) there is a path (which is unique) from the root to each node.

If there is an arc (u, v), u is said to be the *parent* of v and v a *child* of u. The *distance* of a node v in a rooted tree is the length of the path from the root to v.

Given a digraph $G=(V, E)$, a *spanning tree* is a rooted tree $T=(V, S)$ such that $S \subseteq E$. Given a digraph $G=(V, E)$ and a node $x \in V$, a *spanning tree rooted at* x is a tree $X=(T, S)$ rooted at x which satisfies the following properties:

(i) T contains all the nodes which are descendants of x in G;

(ii) $S \subseteq E$.

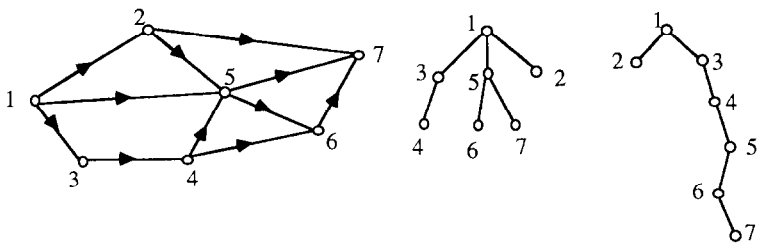

Fig. 1. Trees of minimal paths and maximal paths.

A spanning tree $X=(T,S)$ rooted at x is said to be a *tree of shortest (longest) paths* [14] if there is no spanning tree $X'=(T',S')$ rooted at x such that for each node t

$$distance(t) \text{ in } T' < distance(t) \text{ in } T \ (distance(t) \text{ in } T' > distance(t) \text{ in } T).$$

Figure 1 shows a digraph $G=(V,E)$ with the minimal and maximal length spanning trees rooted at one of its nodes.

3. The data structure

In this section we present a data structure adapted from [13] to maintain a digraph under an arbitrary sequence of add and minpath operations. The basic idea is to represent the transitive closure of $G=(V,E)$. Namely, we associate with each node $x \in V$ a set $desc(x)$ containing all the descendants of x in G. In order to easily extract information about minimal length paths, $desc(x)$ for each $x \in V$ is organized as a minimal length spanning the tree rooted at x.

In addition, while updating these structures during new arc insertions, we would like to access each node in the spanning trees very quickly. To achieve this goal, we make use of a $|V| \times |V|$ matrix of pointers defined as follows:

$$index(i,j) = \begin{cases} \text{points to the node } j \text{ in } desc(i) & \text{if } j \in desc(i), \\ \text{contains a null pointer} & \text{otherwise.} \end{cases}$$

Together with each node in a spanning tree its distance is stored, an information which turns out to be useful while updating minimal spanning trees after an add operation and allows one to perform fast queries about the length of minimal paths between two nodes in O(1) time.

```
function length (nodes: x, y): integer;
begin
  if (index(x, y) ≠ null)
    then  return (distance (index(x, y))
    else  return (+ ∞)
end length;
```

We now give a formal description of the algorithm which maintains the data structure while performing the add and minpath operations. The data structure is initialized at the price of $O(n^2)$ time as follows:

```
procedure initialize;
nodes: x, y;
begin
  for each x∈V do
  begin
    for each y∈V do index(x, y):=null;
    desc(x):=[x] {x is the only node in desc(x)};
    let index(x, x) point to the root of desc(x)
  end
end initialize;
```

A minimal path from i to j can be returned by first examining the entry $index(i, j)$. If it contains a *null* pointer, then there is no path from i to j. Otherwise, $index(i, j)$ allows one to easily locate the position of j in the minimal length spanning tree rooted at i. If reversal pointers to the parent for each node in the spanning trees are maintained, a bottom-up traversal from j to the root i takes at most $O(k)$ time units to return a minimal length path from i to j, where $k<n$ is the number of arcs of the achieved path.

```
procedure path (nodes: x, y; list of nodes: T);
pointer to nodes: p;
begin
  T:=∅
  p:=index(x, y);
  while (p≠null) do
  begin
    insert in T the node pointed by p;
    p:=parent(p)
  end
end path;
```

As far as add operations are concerned, if we denote by $d(x, y)$ the length of a minimal path from x to y in $G=(V, E)$, the following properties must hold:

(P1) $\quad d(x, y)=\min_{u\in V}[d(x, u)+d(u, y)], \quad \forall x, y\in V,$

(P2) $\quad d(x, x)=0, \quad \forall x\in V.$

Furthermore, if $d'(.,.)$ denotes the minimal length function after the insertion of an arc, say the arc (i, j),

(P3) $\quad d'(x, y)=\min[d(x, y), d(x, i)+w(i, j)+d(j, y)], \quad \forall x, y\in V.$

That is, (i, j) can introduce new minimal paths only between ancestors of i ($d(x, i) < +\infty$) and descendants of j ($d(j, y) < +\infty$). In such a case all the spanning trees rooted at x, with x ancestor of i, might be updated taking into account the descendants of j.

```
procedure add (nodes: i, j; integer: w);
node: x;
begin
  for each x∈V do
    if (index (x, i)≠null)   {i.e.: if x is an ancestor of i}
      then paste(x, i, j, w)
end add;
```

The procedure *paste*(x, i, j, w) updates *desc*(x) because of an *add*(i, j, w) operation. This can be accomplished by simply traversing *desc*(x) in a breadth-first manner with the following rules.

(R1) When a node y for which $d(x, y) > d(x, i) + w(i, j) + d(j, y)$ has been reached, then a shorter path between nodes x and y was created by the insertion of the arc (i, j). As a consequence, in *desc*(x) the node y has to be made

- a child of i if $y = j$;
- a child of the same parent it has in *desc*(j) otherwise.

As a consequence, its distance in *desc*(x) must be properly decreased.

(R2) When a node y for which $d(x, y) \leqslant d(x, i) + w(i, j) + d(j, y)$ has been reached, then no shorter path between nodes x and y was created by *add*(i, j, w). The search is no longer prosecuted in the subtree of *desc*(j) rooted at y.

The following lemma gives reason for not prosecuting the search in rule (R2).

Lemma 3.1. *If before an add*(i, j, w) *operation there are two nodes* $u, v \in V$ *for which* $d(u, v) \leqslant d(u, i) + w(i, j) + d(j, v)$ *then* $d(u, w) \leqslant d(u, i) + w(i, j) + d(j, w)$ *for each node* w *in the subtree of desc*(j) *rooted at* v.

Proof. Consider any node w in the subtree of *desc*(j) rooted at v. Owing to property (P1),

$$d(u, w) \leqslant d(u, v) + d(v, w) \leqslant d(u, i) + w(i, j) + d(j, v) + d(v, w).$$

Since w is a descendant of v in the minimal length spanning tree rooted at j, $d(j, w) = d(j, v) + d(v, w)$. This proves the lemma. □

Using a queue in order to implement a breadth-first search [1], the following procedure shows how to update a minimal length spanning tree rooted at x after an add(i, j) operation (see Fig. 2).

```
1.  procedure paste (nodes: x, i, j; integer: w);
2.  nodes: y, w; queue: Q;
3.  begin
4.    Q:=[j];
5.    while Q≠∅ do
6.    begin
7.      y:=dequeue(Q);
8.      if length(x, i)+w(i, j)+length(j, y)<length(x, y)
9.        then begin
10.                 if j=y
11.                 then
12.                   insert y in desc(x) as a child of i
13.                 else
14.                   insert y in desc(x) as a child of parent(index(j, y));
15.                 distance(index(x, y)):=length(x, i)+w(i, j)+length(j, y);
16.                 for each w child of y in desc(j) do
17.                   enqueue(w, Q)
18.               end
19.   end
20. end paste;
```

Fig 2.

4. Correctness and analysis

In this section we begin by proving the correctness of the whole algorithm. Next we analyze its space and the expected time complexity. The correctness hinges on the following theorem.

Theorem 4.1. *At any time, for the data structure the following statement is true:*

$$length(x, y) = d(x, y), \quad \forall x, y \in V.$$

That is, the function length which returns the distance of y in desc(x) is a minimal length function.

Proof. By induction on the number of *add* operations performed.

The basis of the induction is easily proved when no arcs are added since $length(x, y) = +\infty$, $\forall x, y \in V$.

Suppose the thesis true before the insertion of an arc(i, j) of weight $w(i, j)$. Let us call $length'(x, y)$ and $d'(x, y)$ the values of $length(x, y)$ and $d(x, y)$ after inserting such an arc. We want to prove that

$$length'(x, y) = d'(x, y), \quad \forall x, y \in V.$$

The case $length'(x, y) < d'(x, y)$ cannot happen since by Lines 8 and 13 in the procedure *paste*,

$$length'(x, y) \geq \min(length(x, y),\ length(x, i) + w(i, j) + length(j, y))$$
$$= \min(d(x, y),\ d(x, i) + w(i, j) + d(i, j)) = d'(x, y).$$

for the inductive hypothesis.

Suppose that there exists a couple of nodes $x, y \in V$ such that $length'(x, y) > d'(x, y)$. This can happen only if x is an ancestor of i and y a descendant of j and y was not inserted in the queue Q while executing $paste(x, i, j, w)$. As a consequence, there will be a node v which is ancestor of y in $desc(j)$ and in which the procedure *paste* has been stopped on line 8.

Hence, $length(x, v) \leq length(x, i) + w(i, j) + length(j, v)$ or, what is the same for the inductive hypothesis, $d(x, v) \leq d(x, i) + w(i, j) + d(j, v)$.

By applying Lemma 3.1 (since y is in the subtree rooted at v), we also have $d(x, y) \leq d(x, i) + w(i, j) + d(j, y)$ and if we consider property (P3), $d'(x, y) = \min[d(x, y),\ d(x, i) + w(i, j) + d(j, y)] = d(x, y)$.

Thus, $length'(x, y) > d'(x, y) = d(x, y) = length(x, y)$, which is clearly a contradiction since the distance of a node can be only decreased by the procedure *paste*.

Since it can be neither $length'(x, y) < d'(x, y)$ nor $length'(x, y) > d'(x, y)$, the theorem is proved. □

The rest of this section is devoted to the analysis of the complexity of add.

Theorem 4.2. *The worst-case cost of the add procedure is* $O(n^2)$.

Proof. The proof easily follows from the observation that for each node x in V and each node y in $desc(j)$, the algorithm performs a constant number of operations. □

Theorem 4.2 implies that the cost of a sequence of $O(n^2)$ add operations is $O(n^4)$. Hence, our algorithm has the same worst-case performance of the best known bound for the problem.

We now turn our attention to the expected cost and we show that the expected cost of add is $O(n \max(W, \log n))$ amortized over a sequence of $O(n^2)$ operations. Namely, Theorem 4.8 shows that the expected total cost of $n^2 - n$ add operations that, starting from the empty graph with n nodes, allow one to obtain the complete directed graph is $O(n^3 \max(W, \log n))$. We assume that at each step all arcs that do not belong to the graph have the same probability to be inserted. This implies that all the $(n^2 - n)!$ possible sequences of arc insertions have the same probability to occur. We notice that our result improves over the previous known bounds if W, the maximum arc cost, is bounded by n.

The proof of the theorem uses three lemmata:

(i) Lemma 4.4 analyzes the cost of the first $M_1 = c\, n \log n$ add operations, where c is quite a large positive constant (greater than 10 000) to be specified later.

(ii) Lemma 4.5 analyzes the expected cost of the following $M_2 = n^{3/2+\varepsilon} - M_1$ add operations, where ε is a positive constant.

(iii) Lemma 4.7 analyzes the expected cost of the last $n^2 - n - M_2 - M_1$ add operations.

The proofs of Lemmata 4.4, 4.5 and 4.7 are based on the theory of random graphs [5]. We recall that there are two main models that allow one to define random directed graphs. In the first model $D_M(n)$ represents the probability distribution of the directed graphs with n nodes and M arcs; usually, it is assumed that all such graphs are equally likely. In the second model $D(n, p)$ represents the probability distribution of all directed graphs with n nodes obtained by assuming that each arc has probability p to occur, independent of the existence of any other arc. In this model the total number of arcs is the random variable that denotes the total number of successes in $n^2 - n$ Bernoulli trials each with probability p of success. Hence, the expected number of arcs of a graph belonging to $D(n, p)$ is $p(n^2 - n)$ (we do not allow selfloops).

In the proofs of lemmata 4.4, 4.5 and 4.7 we need to characterize the properties of random graphs belonging to the $D_M(n)$ model. On the other hand, it is much easier to prove results in the $D(n, p)$ model, because we can exploit the independence condition on the existence of the arcs. It is not difficult to see that the two models of random graphs are closely related. Informally, for many graph functions, the value of an integer function X defined over a random graph belonging to $D_M(n)$ is very close to the value of the function defined over a random graph belonging to $D(n, p)$, where $p = M/(n^2 - n)$. The next fact gives a sufficient condition that will be exploited throughout the proof of the theorem. Let $N = n^2 - n$ and let $\mathbf{E}_M[X]$ ($\mathbf{E}_p[X]$) denote the expected value of a random variable X in $D_M(n)$ ($D(n, p)$).

Fact 4.3. *Suppose that* $0 < p_2 = p_2(n) < p_1 = p_1(n) < 1$ *and that* $M(n)$ *is an integer function. Assume furthermore that*

$$\lim_{n \to \infty} p_1 q_1 N = \lim_{n \to \infty} p_2 q_2 N = \infty$$

and

$$\lim_{n \to \infty} (p_1 N - M(n))/(p_1 q_1 N)^{1/2} = \lim_{n \to \infty} (M(n) - p_2 N)/(p_2 q_2 N)^{1/2} = \infty,$$

where $q_i = 1 - p_i$, $i = 1, 2$ *and* $N = n^2 - n$.

(i) *If the function* X *is a monotonous increasing positive function (i.e.* $X(G) \leqslant X(H)$ *whenever* $G \subset H$*), then* $\mathbf{E}_M[X] \leqslant \mathbf{E}_{p_1}[X]\,(1 + o(1))$.

(ii) *If the function* X *is a monotonous decreasing positive function (i.e.* $X(G) \geqslant X(H)$ *whenever* $G \subset H$*), then* $\mathbf{E}_M[X] \leqslant \mathbf{E}_{p_2}[X]\,(1 + o(1))$.

Proof. The proof of the fact is a simple modification of the proof for the case of undirected graphs (see e.g. [5, p. 36]). □

Let G_1 and G_2 be now the graphs obtained after the insertion of M_1 and M_1+M_2 arcs, respectively. G_1 and G_2 are random graphs sampled from $D_{M_1}(n)$ and $D_{M_1+M_2}(n)$, respectively.

Lemma 4.4. *The total cost of $M_1 = c\,n\log n$ add operations starting from the empty graph is $O(n^3 \log n)$.*

Proof. It is sufficient to observe that the worst-case cost of each call to insert is $O(n^2)$. □

Lemma 4.5. *The total expected cost of $M_2 = n^{3/2+\varepsilon} - M_1$ add operations starting from G_1 is $o(W n^3)$.*

Proof. Let us define

$$I_{d,s}(h,k) = \begin{cases} 1 & \text{if } (dist(h,k)=d \text{ in } G_1) \text{ and } (outdeg(k)=s \text{ in } G_2), \\ 0 & \text{otherwise.} \end{cases}$$

We first claim that the total cost of M_2 calls to add starting from G_1 is bounded above by

$$\sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=1}^{n}\sum_{s=1}^{n} dsI_{d,s}(h,k). \tag{1}$$

In fact, when the algorithm considers node h and updates the distances from h because of the insertion of a new arc (i, j) it traverses the arc (k, v) if and only if the distance between h and k has been decreased because of the insertion of (i, j). The claim follows from the observation that the distance between h and k in G_1 is d and the outdegree of k in G_2 is s.

We can write (1) as follows:

$$\begin{aligned}
&\sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=1}^{n}\sum_{s=1}^{n} dsI_{d,s}(h,k) \\
&\quad = \sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=1}^{W\log n}\sum_{s=1}^{2n^{1/2+\varepsilon}} dsI_{d,s}(h,k) \\
&\qquad + \sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=W\log n+1}^{n}\sum_{s=1}^{2n^{1/2+\varepsilon}} dsI_{d,s}(h,k) \\
&\qquad + \sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=1}^{n}\sum_{s=2n^{1/2+\varepsilon}+1}^{n} dsI_{d,s}(h,k).
\end{aligned}$$

Now we bound the expected values of the three terms of the right-hand side.

(i) By definition of $I_{d,s}(h,k)$ we have for each h and k

$$\sum_{d=1}^{W\log n}\sum_{s=1}^{2n^{1/2+\varepsilon}} I_{d,s}(h,k)\leqslant 1.$$

Hence, we obtain that, deterministically,

$$\sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=1}^{W\log n}\sum_{s=1}^{2n^{1/2+\varepsilon}} dsI_{d,s}(h,k)\leqslant 2n^{5/2+\varepsilon}W\log n=o(Wn^3).$$

(ii) Let us define the following function:

$$J(h,k)=\begin{cases}1 & \text{if } dist(h,k)>W\log n \text{ in } G_1,\\ 0 & \text{otherwise.}\end{cases}$$

Note that if $outdeg(h)=0$ in G_2, then the algorithm does not consider descendants of node h when updating distances upon the insertion of a new arc. By definition of $I_{d,s}(h,k)$ and $J(h,k)$ we have for each h and k

$$\sum_{d=W\log n+1}^{n}\sum_{s=1}^{2n^{1/2+\varepsilon}} dsI_{d,s}(h,k)\leqslant 2Wn^{3/2+\varepsilon}J(h,k).$$

Hence, we obtain

$$\sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{d=W\log n+1}^{n}\sum_{s=1}^{2n^{1/2+\varepsilon}} dsI_{d,s}(h,k)<2Wn^{5/2+\varepsilon}\left[\sum_{k=1}^{n}J(h,k)\right].$$

In the appendix we show the following fact that completes the proof of case (ii).

Fact 4.6. *The expected value of* $\sum_{k=1}^{n}J(h,k)$ *in* G_1 *is* $O(n^{-1})$.

(iii) Given a graph G, let us define the following function:

$$L_k(G)=\begin{cases}1 & \text{if } outdeg(k)>2n^{1/2+\varepsilon} \text{ in } G,\\ 0 & \text{otherwise.}\end{cases}$$

By definition of $I_{d,s}(h,k)$ and $L_k(G_2)$ we have for each h and k

$$\sum_{d=1}^{n}\sum_{s=2n^{1/2+\varepsilon}}^{n} dsI_{d,s}(h,k)\leqslant n^2L_k(G_2).$$

Hence, we obtain

$$\sum_{h=1}^{n}\sum_{k=1}^{n}\sum_{s=2n^{1/2+\varepsilon}}^{n}\sum_{d=1}^{n} dsI_{d,s}(h,k)<n^3\sum_{k=1}^{n}L_k(G_2).$$

In order to complete the proof it is sufficient to show that the expected value of $\Sigma L_k(G_2)$ is $O(n^{-r})$ for any positive r. In fact, let $\mathbf{E}_{M_2}[L_k]=\mathbf{E}[L_k(G_2)]$ and $\mathbf{E}_{p_3}[L_k]=\mathbf{E}[L_k(G_3)]$, where G_3 is a random graph belonging to $D(n,p_3)$ and

$p_3 = n^{-1/2+2\varepsilon}$. Since L is a monotonous increasing positive function, we can apply Fact 4.3(i) to obtain

$$\mathbf{E}_{M_2}[L_k] = (1+o(1))\mathbf{E}_{p_3}[L_k].$$

Note that L_k in G_3 is Bernoulli-distributed with mean $p_3(n-1)$. Applying Chernoff's bounds [2] on the tail of the binomial distribution we have the probability of $L_k > 2p_3(n-1)$ is less than $e^{-p_3(n-1)/3} < O(n^{-r})$ for any positive r. Hence, $\mathbf{E}_{p_3}[L_k]$ is also $O(n^{-r+1})$ for any positive r.

This completes the proof of case (iii) and of Lemma 4.5. □

Lemma 4.7. *The total expected cost of $n^2-n-M_2-M_1$ add operations starting from G_2 is $O(Wn^3)$.*

Proof. Let (i, j) be the generic arc inserted and let h be a node belonging to the set of ancestors of i. When the algorithm updates the distances from h to all remaining nodes it traverses arc (k, v) if and only if the distance between h and k has been decreased. Hence, for any pair (h, k) the total cost of $n^2-n-M_2-M_1$ calls to add is bounded above by n times the distance between h and k in G_2. Let $\mathbf{E}_{M_2}[dist(h,k)]$ and $\mathbf{E}_{p_4}[dist(h,k)]$ be, respectively, the expected distances between h and k in G_2 and G_4, where G_4 is a random graph belonging to $D(n, p_4)$ and $p_4 = n^{-1/2+\varepsilon/2}$. Since the distance between a pair of nodes is a monotonous decreasing positive function, we can apply Fact 4.3(ii) to obtain

$$\begin{aligned}\mathbf{E}[\text{cost of } n^2-n-M_2-M_1 \text{ calls to add in } G_2] &< \sum_{h=1}^{n}\sum_{k=1}^{n} n\mathbf{E}_{M_2}[dist(h,k)] \\ &< n^3(1+o(1))\mathbf{E}_{p_4}[dist(h,k)] \\ &< 2Wn^3(1+o(1)).\end{aligned}$$

The last inequality follows from the fact (see e.g. [5] for a proof in the undirected case) that in G_4 the expected minimum number of arcs of a path between h and k is $(2+o(1))$; hence, the expected value of $\mathbf{E}_{p_4}[dist(h,k)]$ is less than or equal to $2W$. This completes the proof of Lemma 4.7. □

Theorem 4.8. *If $W < n$, then the total expected cost of n^2-n add operations is $O(n^3 \max(W, \log n))$.*

Proof. The proof follows directly from Lemmata 4.4, 4.5 and 4.7. □

5. Maximal length paths

In this section we extend the results obtained to the case of maximal length paths. In particular, we show how a slight modification of the algorithm proposed is able to perform maxpath operations (i.e. queries about maximal length paths) on dags, where

the insertions of new arcs do not introduce cycles. This is not a significant restriction since the longest path problem is known to be NP-complete for arbitrary graphs [10], while a polynomial algorithm exists for dags [14].

Following the ideas outlined in Section 3, we maintain the descendants of each node as a *maximal* length spanning tree. In order to perform maxpath operations, we simply make the procedure *path* run on these maximal length spanning trees.

Contemporary queries about minimal and maximal paths can be supported by maintaining both minimal and maximal length spanning trees, thus, at the price of duplicating the space required for each node.

It is possible to prove that in case of dags, properties similar to (P1) and (P3) hold. In fact, if we denote by $D(x, y)$ the length of a maximal path from x to y in $G=(V, E)$ ($D(x, y)=-\infty$ if there is no path from x to y), the following must be true:

(P4) $\quad D(x, y)=\max_{u\in V}[D(x, u)+D(u, y)], \quad \forall x, y\in V;$

(P5) $\quad D(x, y)=0, \quad \forall x\in V.$

Furthermore, if $D'(.,.)$ denotes the maximal length function after the insertion of an arc, say the arc (i, j), then

(P6) $\quad D'(x, y)=\max[D(x, y), D(x, i)+w(i, j)+D(j, y)], \quad \forall x, y\in V.$

It is important to underline that property (P4) cannot be extended to the case of digraphs containing cycles.

Consequently, the main results of the previous sections could be extended only in case of directed acyclic graphs. In particular, procedures *add* and *paste* could be applied, mutatis mutandis, for maintaining maximal length spanning trees while inserting new arcs in a dag. As a straightforward modification of Theorem 4.1, the correctness of the algorithm can be guaranteed only when new arc insertions do not introduce any cycle in the original dag.

6. Conclusion

In this paper we have described a fast algorithm for maintaining a directed graph $G=(V, E)$ under an arbitrary sequence of add and minpath operations in graphs with integer edge cost. We proposed a data structure which supports each minpath operation in $O(k)$ worst-case time (where k is the number of arcs in the achieved path) and any number of add operations in a total of $O(\min(n^4, n^3 \max(W, \log n)))$ expected time, where W is the maximum arc weight. As a subsidiary result, also maxpath operations can be performed provided that add operations preserve acyclicity in an original dag. In a forthcoming paper [4], we show how to make a worst-case of $O(n^3 W \log n)$ at the price of doubling the space and of using more sophisticated data structures (and, therefore, likely to be less practical).

There are several related, interesting and perhaps more intriguing problems. First of all, it seems promising to investigate whether there exists an algorithm which is efficient in an amortized sense [21] for both add and minpath operations. Furthermore, given a labelled digraph, is it possible to perform minpath and other more general operations within the same bounds proposed in this paper? Finally, the impact of deletions of arcs in this problem deserves further investigation.

Appendix

Given a graph G and a node h, let $\Gamma_1(h, G_1)$ be the set of nodes of G that are adjacent to h and $\Gamma_d(h, G)$ be the set of nodes at distance d from h in G. Formally, we have

$$\Gamma_1(h, G) = \{ y \mid (h, y) \in E \},$$

$$\Gamma_d(h, G) = \{ y \mid (v, y) \in E,\ v \in \Gamma_{d-1}(h, G),\ y \notin \Gamma_f(h, G)\ f < d \}, \quad d > 1.$$

The proof of Fact 4.6 is based on the analysis of the cardinalities of the sets $\Gamma_d(h, G)$, $d < \log n$. We exploit a beautiful result obtained by Bollobas (see [5]). Namely, in order to characterize the diameter of a random undirected graph, Bollobas gives precise bounds on $\Gamma_d(h, G)$, $d \geqslant 1$, in the case of random undirected graphs. We observe that the extension to the directed case is simple; the following fact is a weaker version of Bollobas' result that is sufficient for our purposes.

Fact A.1. [5, p. 231]. *Let* $d = \lfloor (\log n + \log 0.625)/\log(pn) \rfloor$ *and let* G_p *be a random directed graph with* $p > (10\,000 \log n)/n$. *Then for sufficiently large* n

$$|\Gamma_k(h, G_p)| = (pn)^k (1 \pm \varepsilon)$$

for any $\varepsilon > 0$, *for any vertex* h *and for any* k, $1 \leqslant k \leqslant d-1$, *with probability greater than* $1 - n^{-10}$.

In the sequel we also use the following inequality:

$$(1 - x) \leqslant e^{-x} \quad \text{for all } x.$$

Proof of Fact 4.6. Let $p > 10\,000\ (\log n)/n$. We first specify the value of the constant c of Lemma 4.4; namely, we require $c > pn/(\log n)$. Since $J(h, k)$ is a monotonous decreasing function, by Fact 4.3 the expected value of the sum of $J(h, k)$ in G_1 can be bounded above using the expected value of the sum in G_p. In order to bound this latter quantity let $\Delta(h, G_p)$ be the set of nodes not belonging to $\Gamma_e(h, G_p)$ for $e \leqslant d+1$, where the value d is defined in Fact A.1 and let A be the following event:

$$|\Gamma_d(h, G_p)| < (pn)^d (1 - \varepsilon).$$

Since $J(h, k) < 1$, then, in the same hypothesis of Fact A.1 we have

$$\begin{aligned}\mathbf{E}\left[\sum_{k=1}^{n} J(h,k)\right] &= \mathbf{E}\left[\sum_{k=1}^{n} J(h,k) \mid A\right] \text{Prob}\{A\} \\ &\quad + \mathbf{E}\left[\sum_{k=1}^{n} J(h,k) \mid \textit{not } A\right] \text{Prob}\{\textit{not } A\} \\ &\leqslant n \text{ Prob}(|\Gamma_d(h, G_p)| < (pn)^d(1-\varepsilon)) \\ &\quad + \mathbf{E}[|\Delta(h, G_p)| \mid |\Gamma_d(h, G_p)| \geqslant (pn)^d(1-\varepsilon)] \\ &= \mathrm{O}(n^{-9}) + \mathbf{E}[|\Delta(h, G_p)| \mid |\Gamma_d(h, G_p)| \geqslant (pn)^d(1-\varepsilon)].\end{aligned}$$

Hence, to complete the proof it is sufficient to show that if Fact A.1 holds, then

$$\mathbf{E}[|\Delta(h, G_p)| \mid |\Gamma_d(h, G_p)| \geqslant (pn)^d(1-\varepsilon)] = \mathrm{O}(n^{-1}).$$

Easy calculations show that $d+2 < \log n$ and that for sufficiently large n

$$(pn)^d \geqslant 0.625/p.$$

We distinguish two cases depending on the cardinality of $s = |\Gamma_d(h, G)|$.

Case 1: $s \geqslant n/2$. If $s \geqslant n/2$, then $|\Delta(h, G_p)|$ is less than the expected number of nodes not connected to $\Gamma_{d+1}(h, G_p)$. This quantity has binomial distribution. More precisely the generic node x is not connected to a node of $\Gamma_{d+1}(h, G_p)$ with probability $(1-p)$; hence, x is not connected to any node of $\Gamma_{d+1}(h, G_p)$ with probability

$$(1-p)|\Gamma_{d+1}(h, G_p)| \leqslant (1-p)^{n/2}.$$

Hence, the expected number of nodes not connected to $\Gamma_{d+1}(h, G_p)$ is less than

$$n(1-p)^{n/2} \leqslant n\,\mathrm{e}^{-pn/2} = \mathrm{O}(n^{-1}).$$

Case 2: $s < n/2$. If $0.625/p \leqslant s < n/2$, then we first show that $\Gamma_{d+1}(h, G_p)$ is greater than $0.4n$ with overwhelming probability. In fact, the number of nodes belonging to $\Gamma_{d+1}(h, G_p)$ has binomial distribution with parameters p' and n' and expected value $n'p'$. The probability of success of a trial, p', is greater than 0.438; in fact, $x \notin \Gamma_{d+1}(h, G_p)$ with probability $(1-p)^{|\Gamma_{d+1}(h, G_p)|} \leqslant \mathrm{e}^{-ps}$; hence, $x \in \Gamma_{d+1}(h, G_p)$ with probability p' greater than $(1-\mathrm{e}^{-ps}) \leqslant 0.438$. On the other hand, we have

$$n' = n - \sum_{f \leqslant d} |\Gamma_f(h, G_p)|.$$

If Fact A.1 holds, then n' can be bounded as follows:

$$\begin{aligned}n' &= n - \sum_{f \leqslant d} |\Gamma_f(h, G_p)| > n - \sum_{f \leqslant d} (pn)^f(1+\varepsilon) \\ &= n - (1+\varepsilon)(pn)^d \sum_{f \leqslant d} (pn)^{-f}.\end{aligned}$$

Note that the sum in the last term of the above formula, for sufficiently large n, converges to a value less than 1.001. Since Fact A.1 holds for any ε, we have

$$n' > n - 1.01s.$$

Hence, we have shown that if Fact A.1 holds then $n'p'$, the expected number of nodes belonging to $\Gamma_{d+1}(h, G_p)$, is greater than

$$0.438\ (n - 1.01s).$$

In order to prove that the cardinality of $\Gamma_{d+1}(h, G_p)$ is greater than 0.4 with overwhelming probability we apply Chernoff's bounds to the tail of the binomial distribution (see e.g. [2]). More precisely, let β be a suitable positive constant less than 1; the cardinality of $\Gamma_{d+1}(h, G_p)$ is greater than $(1-\beta)n'p'$ with probability greater than

$$1 - e^{-\beta^2 p' n'/3} \leqslant 1 - e^{-\beta^2(1-e^{-ps})n'/3}.$$

Let us define the following function

$$f(s) = (1-\beta)(1-e^{-ps})(n - 1.01s).$$

Since $(1-e^{-ps})(n-1.01s)$ is less than $n'p'$, we have that, if Fact A.1 holds, then the cardinality of $\Gamma_{d+1}(h, G_p)$ is greater than $f(s)$ with probability greater than

$$1 - e^{-\beta^2(1-e^{-ps})(n-1.01s)/3} > 1 - n^{-t} \quad \text{for any } t > 0.$$

Note that $f(s)$ is concave and has its minimum for $s = 0.625/p$. We have for sufficiently large n and sufficiently small β

$$f(0.625/p) = (1-0.438)(1-\beta)(n-1.01s) \geqslant 0.4n.$$

As in case 1 the last step is to show that the expected number of nodes not connected to $\Gamma_{d+1}(h, G_p)$ is sufficiently small. In fact, we have that a node is not connected to a node belonging to $\Gamma_{d+1}(h, G_p)$ with probability less than $(1-p)^{f(s)}$; hence, the expected number of nodes not connected to a node belonging to $\Gamma_{d+1}(h, G_p)$ is less than

$$n(1-p)^{f(s)} < ne^{-pf(s)} < O(n^{-1}). \qquad \square$$

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] D. Angluin and L.G. Valiant, Fast probabilistic algorithms for Hamilton circuits and matching, *J. Comput. System Sci.* **18** (1979) 155–193.

[3] G. Ausiello, A. D'Atri and M. Moscarini, Chordality properties on graphs and minimal conceptual connections in semantic data models, *J. Comput. System Sci.* **33** (2) (1986) 179–202.

[4] G. Ausiello, G.F. Italiano, A. Marchetti Spaccamela and U. Nanni, Incremental algorithms for minimal length paths, manuscript.

[5] B. Bollobas, *Random graphs* (Academic Press, London, 1985).

[6] S. Even and H. Gazit, Updating distances in dynamic graphs, *Methods Oper. Res.* **49** (1985) 371–387.

[7] S. Even and Y. Shiloach, An on-line edge deletion problem, *J. ACM*, **28** (1981) 1–4.
[8] H. Frank and I.T. Frish, *Communication, Transmission and Transportation Networks* (Addison-Wesley, Reading, MA, 1971).
[9] G.N. Frederickson, Data structures for on-line updating of minimum spanning trees with applications, *SIAM J. Comput.*, **14** (1985) 781–798.
[10] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness* (Freeman, San Francisco, CA, 1979).
[11] F. Harary, *Graph Theory* (Addison-Wesley, Reading, MA, 1972).
[12] T. Ibaraki and N. Katoh, On-line computation of transitive closure of graphs, *Inform. Process. Lett.* **16** (1983) 95–97.
[13] G.F. Italiano, Amortized efficiency of a path retrieval data structure, *Theoret. Comput. Sci.* **48** (1986) 273–281.
[14] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids* (Holt, Rinehart and Winston, New York, 1976).
[15] A. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected running time $O(n^2 \log n)$, in: *Proc. 26th Annual Symp. on Foundations of Computer Science* (1985) 101–105.
[16] N.J. Nilsson, *Problem Solving Methods in Artificial Intelligence* (McGraw-Hill, New York, 1971).
[17] H. Rohnert, A dynamization of the all pairs least cost path problem, in: *Proc. 2nd Symp. on Theoretical Aspects of Computer Science* (1985) 279–286.
[18] D.D. Sleator and R.E. Tarjan, A data structure for dynamic trees, in: *Proc. 13th Symp. on Theory of Computing* (1981) 114–122.
[19] J.F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine* (Addison-Wesley, Reading, MA, 1984).
[20] R.E. Tarjan, *Data Structures and Network Algorithms* (Society of Industrial and Applied Mathematics, Philadelphia, PA, 1983).
[21] R.E. Tarjan, Amortized computational complexity, *SIAM J. Algebraic Discrete Methods* **6** (1985) 306–318.